BUILT-IN RESEEDING FOR SERIAL BIST

Ahmad A. Al-Yamani and Edward J. McCluskey
Center for Reliable Computing
Stanford University, Stanford, CA
{aaa, ejm}@crc.stanford.edu

Abstract

Reseeding is used to improve fault coverage in BIST pseudo-random testing. Most of the work done on reseeding is based on storing the seeds in an external tester. Besides its high cost, testing using automatic test equipment (ATE) makes it hard to test the circuit while in the system. In this paper, we present a technique for built-in reseeding. Our technique requires no storage for the seeds. The seeds are encoded in hardware. The seeds we use are deterministic so 100% fault coverage can be achieved. Our technique causes no performance overhead and does not change the original circuit under test. Also, the technique we present is applicable for transition faults as well as single-stuck-at faults. Built-in reseeding is based on expanding every seed to as many ATPG patterns as possible. This is different from many existing reseeding techniques that expand every seed into a single ATPG pattern. This paper presents the built-in reseeding algorithm together with a hardware synthesis algorithm and implementation.

1. Introduction

An advantage of built-in self-test (BIST) is its low cost compared to external testing using automatic test equipment (ATE). In BIST, on-chip circuitry is included to provide test vectors and to analyze output responses. One possible approach for BIST is pseudo-random testing using a linear feedback shift register (LFSR) [McCluskey 85]. Among the other advantages of BIST is its applicability while the circuit is in the system.

Many digital circuits contain random-pattern-resistant (r.p.r.) faults that limit the coverage of pseudo-random testing [Eichelberger 83]. The r.p.r. faults are faults with low detectability (few patterns detect them). Several techniques have been suggested for enhancing the fault coverage achieved with BIST. These techniques can be classified as: (1) Modifying the circuit under test (CUT) by test point insertion [Eichelberger 83, Touba 96] or by redesigning the CUT, (2) *Weighted pseudo-random patterns*, where the random patterns are biased using extra logic to increase the probability of detecting r.p.r. faults [Eichelberger 89, Wunderlich 90] and (3) *Mixed-mode testing* where the circuit is tested in two phases. In the first phase, pseudo-random patterns are applied. In the second phase, deterministic patterns are applied to target the undetected faults [Koenemann 91, Hellebrand 95, Touba 00]. We present a mixed mode technique based on inserting deterministic patterns between the pseudo-random patterns.

Modifying the CUT is often not desirable because of performance issues or intellectual property reasons. Weighted pseudo-random sequences require multiple weight sets that are typically stored on chip. Mixed mode testing is done in several ways; one way is to apply the deterministic patterns from an external tester or store them in an on-chip ROM. Additional circuitry is required to apply the patterns in the ROM to the circuit under test. Instead of storing patterns, seeds can be stored on the tester or in the on-chip ROM. These seeds are transferred into the LFSR and then expanded into the scan chains. This technique does not eliminate the need for the circuitry that transfers the seeds from the ROM to the LFSR.

Another technique for mixed-mode testing uses *mapping logic* [Touba 95]. The strategy is to identify patterns in the pseudorandom sequence that don't detect any new faults and map them by hardware into deterministic patterns.

In this paper, we present a technique that combines mapping logic and *reseeding* (loading the LFSR with a new seed). Our technique uses a simple circuit to identify the states at which the LFSR is to be reseeded. It uses minimal additional hardware to choose the new seed. Our technique utilizes the LFSR flip-flops for storing the seeds. The output of the circuit that detects when to reseed is used to pick the new seed by inverting the flip-flops in which the seed is different from the current contents of the LFSR.

Touba's mapping logic alters the outputs of the pseudo-random pattern generator (LFSR) to insert deterministic patterns into the test set. On the other hand, our technique alters the contents of the LFSR to insert seeds, which produce the deterministic patterns.

Our technique can achieve 100% fault coverage while eliminating the need for external testing or for a ROM to store the seeds. Furthermore, it requires fewer seeds to be encoded compared to previous work. Actually, at best, our technique causes an order of magnitude reduction in the number of seeds to be encoded compared to previous work and even in its worst case it needs fewer seeds than previous work. With a small modification, our technique can be applied for transition faults rather than single stuck faults. Even if it's more desirable to apply the deterministic patterns externally than to add the mapping logic, our technique is still applicable and it reduces the seed storage because of the pseudorandom patterns applied between the seeds. The price for this storage reduction is paid in test length.

Our contributions in this paper are summarized as follows: (1) A reseeding technique that eliminates completely the need for external pattern storage or an on-chip ROM. It's based on encoding the seeds in hardware and using special hardware for the LFSR. The technique can be used for both transition faults as well as single-stuck-at faults. (2) A hardware implementation for the given technique. (3) A reseeding algorithm based on the hardware implementation explained in Sec. 4. The algorithm allows the user to trade off test length for hardware overhead.

In Sec. 2 of this paper, we review the related literature. In Sec. 3, we explain the reseeding circuitry implementation. Section 4 discusses the reseeding algorithm. Section 5 shows the simulation results and Sec. 6 concludes the paper.

2. Related work

In serial BIST (aka test per scan), deterministic patterns are encoded into smaller vectors (aka seeds) that are loaded into the LFSR and then expanded into the desired patterns in the scan chains. The patterns are encoded into seeds by solving a linear system of equations, which is an algebraic representation of the linear expansion of the LFSR into the scan chain flip-flops. There are some linear dependencies between some flip-flops of the scan chain. Due to these dependencies, solving the linear system of equations may not always be possible.

2.1 Seed calculation and seed storage

Based on the statistical treatment of the linear dependencies in [Bardell 87], Konemann presented a technique for coding test patterns into LFSRs of size S_{max}+20, where S_{max} is the maximum number of specified bits in the ATPG patterns. By adding 20 bits to S_{max} as the size of the LFSR, the probability of linear dependence drops to 1 in a million [Koenemann 91].

[Rajski 98] presented a reseeding-based technique that improves the encoding efficiency by using variable-length seeds together with a multiple polynomial LFSR. The technique reuses part of the scan chain flip-flops in expanding the seeds.

In [Krishna 01], a new form of reseeding was described for high encoding efficiency. By making use of the degrees of freedom in solving the linear system of equations the paper achieves higher encoding efficiency than static reseeding.

The technique presented in this paper encodes the seeds in hardware instead of storing them in a ROM or in the tester. Other than the circuit needed to detect when to reseed, minimal hardware is needed to load the desired seeds. Also, our technique is orthogonal to all of the above techniques; i.e., it can be combined with partial dynamic reseeding for a high encoding efficiency or with Rajski's technique that utilizes part of the scan chain.

2.2 Hardware-based reseeding

[Savir 90] presented a reseeding scheme that requires having shadow flip-flops for the LFSR flips-flops. The shadow flip-flops contain the next seed. These shadow flip-flops are loaded with the XOR of the old shadow contents and the original LFSR flip-flops contents. This way, the new seed is expected to be far in the sequence from the current contents of the LFSR. Kim presented a method for generating non-successive pseudo-random test patterns by cascading the LFSR with the scan chain and including a feedback from the scan-out signal into the LFSR [Kim 96]. In [Crouch 95], a self re-seeding LFSR was presented. Again the LFSR is loaded with a random seed every time a pattern is repeated in part of the LFSR.

The above schemes have the advantage of diversity of the sequences from which the patterns are drawn. They also have the advantage of not requiring seed storage. However, they use seeds that don't particularly target undetected faults. Our technique is based on deterministic seeds which expand into ATPG patterns so 100% fault coverage can be achieved. We pay the price in hardware overhead.

2.3 Mapping logic

Touba and McCluskey came up with an innovative approach for applying deterministic patterns through mapping logic [Touba 95]. In their technique, random patterns that don't detect r.p.r. faults are mapped to ATPG generated cubes through combinational logic. The mapping is performed in two phases, the source patterns are identified in the first phase, and the ATPG cubes are loaded in the second phase. Several iterative minimization heuristics are applied to reduce the area overhead of the mapping logic.

Our technique is a generalization of mapping logic. It has the following advantages: (1) Mapping logic needs hardware for all patterns. In built-in reseeding the LFSR runs in autonomous mode after loading every seed detecting more r.p.r. faults without having to perform more mappings. Because of this, some patterns may not be required. (2) In mapping logic, we need logic for detecting the pattern to be mapped and more logic to perform the mapping. On the other hand, in our technique we only need the logic that detects the patterns that need to be mapped. Enforcing the new values in the LFSR is done utilizing the current contents of the flip-flops of the LFSR.

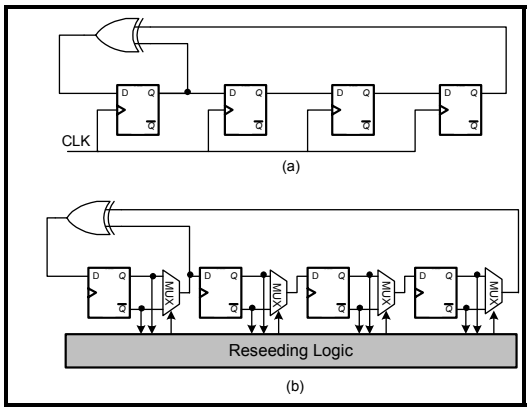

Figure 1: Reseeding circuit connection to LFSR: (a) A standard 4-stage LFSR (b) 4-stage LFSR with reseeding circuit.

3. Reseeding circuitry implementation

The operation of the reseeding circuit is as follows: the LFSR starts running in autonomous mode for some time according to the algorithm described in Sec. 4. Once it is time for reseeding, a seed is loaded into the LFSR, which then goes back to the autonomous mode and so on and so forth until the desired coverage is achieved. The new seed is loaded by putting the LFSR in the state that precedes the seed value, so that at the next clock pulse, the new seed is in the LFSR.

Figure 1(b) shows the structure of the LFSR and its interaction with the reseeding circuit. For our technique, we use muxed flip-flops as shown in the figure. These flip-flops are just like the scan chain flip-flops. By activating the select line of a given mux, the logic value in the corresponding LFSR stage is inverted.

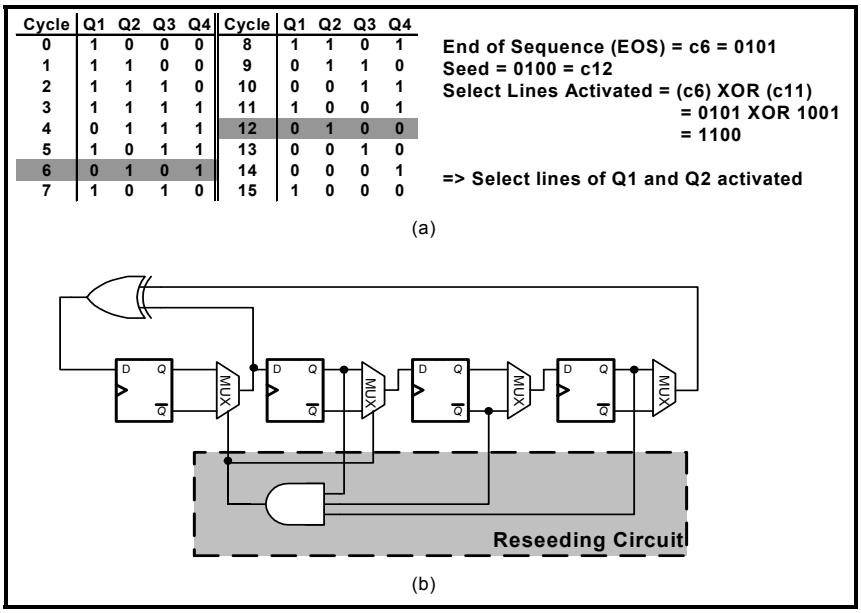

Figure 2: Example reseeding circuit (a) Select lines computation (b) Hardware implementation.

Definition: The *end of sequence* (*EOS*) contents are the values of the LFSR flip-flops before reseeding. The output of the reseeding circuit activates the select lines of the muxes to invert certain stages of the LFSR such that the desired seed is loaded in the next clock cycle.

As seen in the figure, the only modification to the LFSR compared to a regular LFSR is that the LFSR flip-flops are replaced by muxed flip-flops just like the scan chain.

Let's turn our attention to the reseeding circuit by looking at the following example. Figure 2 is an example using a 4-stage *self-reseeding LFSR* (LFSR with reseeding logic) with a primitive polynomial. The table in part (a) shows the full sequence of the regular LFSR. Assume that we want to reseed after the 6th cycle (c6). The reseeding circuit needs to be an AND gate that takes as inputs the contents of the LFSR at c6. So in the example the input to the reseeding AND is $\overline{Q}_1 Q_2 \overline{Q}_3 Q_4$. All the cycles that are not part of the desired sequence can be used to minimize the reseeding circuit.

As an example, let the seed be 0100 (c12); we can easily calculate c11 given the polynomial of the LFSR (c11 = 1001). The reason we calculate c11 and not c12 is because we want the seed to be loaded into the LFSR in the next clock cycle. XORing c6 with c11 yields 1100 which means that the output of the reseeding AND should activate the select lines of the MUXes of Q_1 and Q_2. The truth table for the reseeding circuit is shown in Table 1, where Q_i comes from the output of the i^{th} stage and S_j is the select line of the mux of the j^{th} stage. The patterns between c6 and c11 will not occur so they are don't cares. All the other patterns don't activate any muxes.

The resulting circuit for the example in Figure 2 is a 3-input AND as shown in the figure.

As more seeds are required, every select line will be a function of the end-of-sequence patterns that will activate it to complement the contents of its corresponding flip-flop. We can then optimize the circuit for that select line and multiple-output optimization can be done for all select lines.

Table 1: Table of Combinations for the Reseeding Circuit Example

$Q_1Q_2Q_3Q_4$	$S_1S_2S_3S_4$	$Q_1Q_2Q_3Q_4$	$S_1S_2S_3S_4$
1 0 0 0	0 0 0 0	1 1 0 1	d d d d
1 1 0 0	0 0 0 0	0 1 1 0	d d d d
1 1 1 0	0 0 0 0	0 0 1 1	d d d d
1 1 1 1	0 0 0 0	1 0 0 1	0 0 0 0
0 1 1 1	0 0 0 0	0 1 0 0	0 0 0 0
1 0 1 1	0 0 0 0	0 0 1 0	0 0 0 0
0 1 0 1	1 1 0 0	0 0 0 1	0 0 0 0
1 0 1 0	d d d d	0 0 0 0	d d d d

There are two ways to apply transition fault test sets to circuits with scan chains. One way is to use pairs of clock pulses (launch on capture). Once the 1st pattern is loaded into the scan chain, a clock pulse is applied so the response of the combinational logic to the 1st pattern is latched into the flip-flops. Another clock pulse is then applied such that the response of the combinational logic to the 1st pattern is used as the 2nd pattern in the transition fault pair. The other way is to load the scan chain with the two successive patterns (launch on shift). Once the first pattern is applied, the contents of the scan chain are shifted, the 2nd pattern is applied and the results are captured in the scan chain to be shifted out.

We use the first technique (launch on capture) for transition faults. The reseeding circuit only needs to load the LFSR with the 1st pattern because the 2nd pattern is the response of the logic to the 1st pattern. The reseeding circuit is synthesized such that it changes the contents of the LFSR from the current values to the 1st pattern in the transition fault pattern pair. This means that our technique requires no extra hardware to apply it to transition faults. Although pairs of 2 vectors are required for transition faults, only the first pattern needs to be encoded in hardware because the 2nd pattern is the circuit's response to the 1st pattern.

Figure 3 shows where the reseeding circuit fits in a system level view of a circuit with an LBIST controller. The pattern counter is part of the LBIST controller and it is used to count the patterns applied to the circuit under test (CUT).

Although the experiments are given for a single chain per circuit, the technique we present is directly applicable with multiple chains with any phase shifters.

In a BIST environment, where the LFSR is known in advance and the initial seed and test length are also known, the reseeding circuit may take its inputs from the pattern counter instead of the LFSR contents. The dashed line in Figure 3 corresponds to the reseeding circuit taking its inputs from the pattern counter. If a set of test length TL is applied to the circuit the pattern counter will be of size $\lceil \log_2(TL) \rceil$. If the size of the pattern counter is much smaller than the LFSR, this can lead to large reduction in the complexity of the reseeding circuit because the number of inputs is reduced.

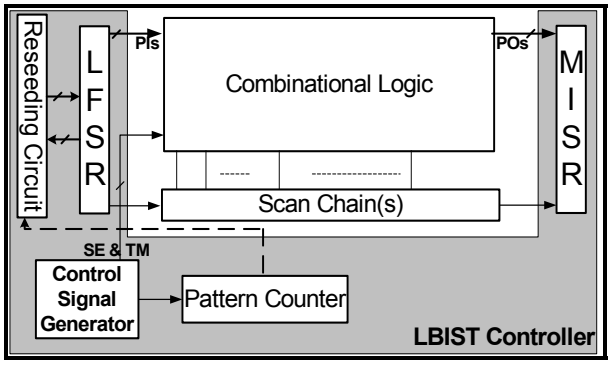

Figure 3: Reseeding circuit in a system view of BIST environment.

4. Reseeding algorithm

The algorithm we present is based on the following strategies: (1) Generate ATPG patterns for faults that were not detected with pseudo-random patterns and calculate the seeds for these patterns, (2) When a seed is loaded into the LFSR, let the LFSR run in autonomous mode for sometime because there is a chance that some pseudo-random patterns will drop more faults so that some of the ATPG patterns are not needed. This may not be a wise idea if the seeds are applied from a tester because applying the pseudorandom patterns after every deterministic pattern takes extra time on the tester where test time may be expensive. On the other hand, if the seeds are stored or coded on-chip, as it is the case in this work, then it is definitely worth it to minimize the number of seeds that need to be loaded for the same coverage. This will directly minimize the area overhead on the chip, (3) As long as the pseudo-random patterns are detecting more faults, the LFSR should continue in pseudorandom mode. When the pseudo-random patterns become ineffective, the LFSR should be loaded with a new pattern. How to quantify the effectiveness of pseudo-random patterns? The answer is in the next paragraphs.

Definition: *coverage improvement threshold* (CIT) is the improvement in fault coverage required by the algorithm to continue applying pseudorandom patterns. It's a parameter to quantify the effectiveness of pseudorandom patterns. As long as applying more pseudo-random patterns improves the coverage by at least CIT%, the pseudo-random patterns will be considered effective. When the improvement falls below CIT%, the pseudo-random patterns are considered ineffective. We need to determine how many patterns are simulated before measuring the improvement in coverage. **Definition:** We define the *step size* as the number of patterns simulated before measuring the improvement in coverage. In our simulations, we used different values for the step size.

The only user-specified parameters for this algorithm are the coverage improvement threshold (CIT) and the step size. At one extreme, choosing a very small CIT, means that the user prefers to stick to pseudo-random patterns as long as they have any improvement in the coverage. This in turn means the user wants low hardware overhead for the reseeding circuit. In return for that, the user is willing to have a long test length. At the other extreme, if the user specifies a very high CIT, it means that he has enough area on the chip for the reseeding circuit.

Reseeding based on CIT is one way to choose when to reseed the LFSR. Many other strategies can be used for selecting the reseeding cycles. One way is to choose a fixed length for running the LFSR in autonomous mode after the seed is loaded.

In a sense, our built-in reseeding algorithm is a generalization of mapping logic. If we choose not to continue running the LFSR in pseudorandom mode after reseeding and we choose to alter the LFSR output of the LFSR contents, then our technique becomes similar to mapping logic. We chose to expand the seeds into many patterns since this should reduce the number of seeds to be loaded and accordingly reduce the area of the reseeding circuit. We also chose to alter the contents of the LFSR to enable a different sequence of pseudorandom patterns to drop more faults.

For some circuits, all we need to catch the undetected faults is to take the LFSR to another location in the pattern space. In that case, one or two seeds are enough. For other circuits, the undetected faults require many seeds to be loaded. In that case, our technique will require many seeds and have a long test length.

In case of transition faults, the only change to the algorithm is that fault simulation and ATPG should be done for transition faults instead of SSFs.

5. Simulation results

We performed our experiments on some ISCAS 89 benchmarks. The characteristics of the benchmarks we used are shown in Table 2. The table shows the number of primary inputs, primary outputs and flip-flops in the circuits. It also shows the size of the LFSR used. The last column shows the cell-area of the circuits in LSI library cells area units. The library used for technology mapping is LSI Logic G.Flex library, which is a 0.13 µ technology library.

5.1 Comparison with previous work

We performed some simulation experiments to compare our built-in reseeding with previous work. Most of the previous work assumes one seed per pattern. In mapping logic, hardware is needed to map each pattern but no pattern or seed

storage is needed. From here on, "previous work" refers to all work that assumes one seed per pattern.

Table 2: ISCAS 89 CUTs Used in The Experiments.

Circuit Name	PIs	POs	Scan Chain Size	LFSR Size	Area
s1423	17	5	74	50	4,531
s1488	8	19	6	6	3,555
s1494	8	19	6	6	3,563
s5378	35	49	179	61	14,376
s9234	36	39	211	80	25,840
s13207	62	152	638	45	44,255
s15850	77	150	534	150	48,494
s35932	35	320	1728	60	106,198
s38417	28	106	1636	200	120,180
s38584	38	304	1426	200	115,855

We compare the number of seeds that need to be encoded if our built-in reseeding technique is used and the number of seeds that need to be stored or mapped if previous techniques are used. The number of seeds determines the area of the reseeding or mapping circuit. Since further area minimization heuristics can be applied to all techniques, it's fair to compare them in terms of the number of seeds that need to be mapped or stored for the same coverage. This comparison can be done for all possible combinations of coverage improvement threshold (CIT) and step size, see Sec. 4. Since built-in reseeding may require longer test length than previous techniques, it's fair to show the test length in the comparison. Why should we tolerate this increase in test length? (1) The area is a scarce resource in BIST. (2) The effect of increasing the test length is not as severe with BIST as it is with external testing because BIST is much faster than external testing. (3) Increasing the test length increases the number of pseudo-random patterns that are likely to be effective in catching unmodeled defects. (4) The increase in test length we have is much smaller than that if only pseudorandom patterns were used. The user can minimize the test length for area overhead.

Table 3 shows a comparison of previous techniques and our reseeding technique. The table is for 100% single-stuck-at fault coverage, 1% CIT (coverage improvement threshold) and 1024 patterns as a step size. In its best case, built-in reseeding reduces the number of seeds required for 100% coverage by an order of magnitude and increases the test length by only a factor of 1.7 compared to using one seed per pattern.

The table shows that there are many cases where reseeding offers a considerable improvement in the number of patterns required to test the circuit and at the same time doesn't cause a large increase in the test length.

5.2 Area overhead and test length for SSFs

In this section we present the area overhead of our built-in reseeding technique using the SSF model. Table 4 shows the area overhead of the reseeding circuits for the benchmarks we used. The reseeding circuits given were designed based on 100% SSF fault coverage. The area overhead given in the table is the minimum area overhead we could achieve for 100% coverage after trying multiple CITs and step sizes.

For most of the circuits, the area overhead ranged from 0.05% to 4%. In a few cases we had a large reseeding circuit.

The minimum area overhead was achieved with different values for CIT for different circuits. This is due to the fact that the number of seeds required to be loaded is highly dependent on the order in which seeds are picked. Our algorithm picks the seeds with the objective of minimizing the area overhead so this might have different effects on the test length.

An important conclusion from the above results is that we need an algorithm to efficiently find the best order for loading the seeds such that the area of the reseeding circuit is minimized. This is investigated in a different paper and results are available in [Alyamani 03].

5.3 Area overhead and test length for transition faults

In this section we present the area overhead of our built-in reseeding technique using the transition fault model. The reseeding circuits given were designed based on the maximum achievable transition fault coverage. For most of the cases, the area overhead ranged from 0.2% to 5% as shown in Table 5.

Table 3: Comparison of Built-In Reseeding and Previous Work Encoding One Seed per Pattern.

Circuit	Number of seeds to be encoded			Test Length		
	One seed per pattern	Built-in reseeding	%Reduction	One seed per pattern	Built-in reseeding	Degradation factor
s1423	7	4	42.9	2,000	8,160	4.1
s1488	4	1	75.0	3,072	4,096	1.3
s1494	4	1	75.0	3,072	4,096	1.3
s5378	60	20	66.7	3,072	27,648	9.0
s9234	93	62	33.3	5,120	76,800	15.0
s13207	121	26	78.5	5,120	60,416	11.8
s15850	41	38	7.3	4,096	45,056	11.0
s35932	18	1	94.4	3,072	5,120	1.7
s38417	78	62	20.5	5,120	89,088	17.4
s38584	107	36	66.4	4,096	76,800	18.8

Table 4: Area Overhead and Test Length For Built-In Reseeding (SSFs)

Circuit	Area	Reseeding Area	% Area Overhead	TL
s1423	4,531	113.2	2.5	8,160
s1488	3,555	12	0.3	6,549
s1494	3,563	12	0.3	6,560
s5378	14,376	462	3.2	928
s9234	25,840	3566	13.8	4,544
s13207	44,255	122	0.3	1,600
s15850	48,494	1988	4.1	2,432
s35932	106,198	52	0.05	6,144
s38417	120,180	9418	7.8	6,016
s38584	115,855	1760	1.5	61,440

In most cases, the minimum area overhead is achieved with different step sizes. This is again due to the fact that the number of seeds that need to be loaded is highly dependent on the order in which seeds are picked.

In general the area overhead of the built-in reseeding circuit for transition faults was close to that for SSFs. The reason is that we only encode the 1st patterns of the transitions patterns pairs in the reseeding circuit. We measure the test length of transition fault test sets in pairs of patterns.

Table 5: Area Overhead and Test Length For Built-In Reseeding (Transition Faults)

Circuit	Area	Reseeding Area	% Area Overhead	TL
s1423	4,531	205.2	4.5	18,432
s1488	3,555	35.2	1.0	3,072
s1494	3,563	45.2	1.3	3,072
s5378	14,376	855.2	5.9	7,680
s9234	25,840	3937.6	15.2	15,904
s13207	44,255	225.2	0.5	7,168
s15850	48,494	2513.6	5.2	7,424
s35932	106,198	174	0.2	1,376
s38417	120,180	3022.8	2.5	34,816
s38584	115,855	2668.4	2.3	33,792

6. Summary and Conclusions

We presented a built-in reseeding scheme based on encoding the seeds in an on-chip reseeding circuit. 100% fault coverage can be achieved with our technique without any external testing. Our built-in reseeding scheme allows the designer to trade-off area overhead for test length in an optimized way.

Our technique uses special hardware for the LFSR such that the reseeding circuitry area overhead is minimized. Also, the technique we presented is directly applicable to the transition fault model. The simulation experiments show that the average area overhead is less than 4% for 100% SSF as well as transition fault coverage.

We also presented a reseeding algorithm that minimizes the area overhead. The algorithm takes care of seed selection and reseeding cycle selection. The simulation results show that, for most of the cases, the test length doesn't have to be maximized when the area overhead is minimized, which is a double win for our technique.

Acknowledgement

This work was supported by King Fahd University of Petroleum and Minerals and by LSI Logic under contract No. 16517.

References

[Alyamani 03] Alyamani, A., S. Mitra and E. J. McCluskey, "BIST Reseeding with Very Few Seeds" *VLSI Test Symposium*, Apr. 2003.

[Bardell 87] Bardell, P.H., W. McAnney, and J. Savir, "Built-In Test for VLSI", John Wiley, New York, 1987.

[Crouch 95] Crouch, Alfred, and M. D. Pressly, "Self Re-seeding Linear Feedback Shift Register Data Processing System for Generating a Pseudo-random Test Bit Stream and Method of Operation," US Patent 5,383,143, Jan. 1995.

[Eichelberger 83] Eichelberger, E. B., and E. Lindbloom, "Random-Pattern Coverage Enhancement and Diagnosis for LSSD Logic Self-Test", *IBM Journal of Research and Development*, Vol. 27, No. 3, pp. 265-272, May 1983.

[Eichelberger 89] Eichelberger, E. B., E. Lindbloom, F. Motica, and J. Waicukauski, "Weigted Random Pattern Testing Apparatus and Method," US Patent 4,801,870, Jan. 89.

[Hellebrand 95] Hellebrand, S., J. Rajski, S. Tarnick, S. Venkataraman and B. Courtois, "Built-in Test for Circuits with Scan Based on Reseeding of Multiple-Polynomial Linear Feedback Shift Registers," *IEEE Transactions on Computers*, Vol. 44, No. 2, pp. 223-233, Feb. 1995.

[Kim 96] Kim, Kee, "Scan-Based Built-In Self Test (BIST) with Automatic Reseeding of Pattern Generator," US Patent 5,574,733, Nov. 1996.

[Koenemann 91] Koenemann, B., "LFSR-Coded Test Patterns for Scan Designs," *Proc. of ETC*, pp. 237-242, 1991.

[Krishna 01] Krishna, C. V., A. Jas, and N. Touba, "Test Vector Encoding Using Partial LFSR Reseeding" *Proc. of International Test Conference*, pp. 885-893, 2001.

[McCluskey 85] McCluskey, E.J., "Built-In Self-Test Techniques," *IEEE Des. & Test of Comp.,* pp. 21-28, Apr. 85.

[Rajski 98] Rajski, J., J. Tyszer and N. Zacharia , "Test Data Decompression for Multiple Scan Designs with Boundary Scan," *IEEE Transactions on Computers*, Vol. 47, No. 11, pp. 1188-1200, Nov. 1998.

[Savir 90] Savir, J., and W. McAnney, "A Multiple Seed Linear Feedback Shift Register," *Proc. of International Test Conference*, pp. 657-659, 1990.

[Touba 95] Touba, N.A., and E.J. McCluskey, "Synthesis of Mapping Logic for Generating Transformed Pseudo-Random Patterns for BIST," *Proc. of International Test Conference*, pp. 674-682, 1995.

[Touba 96] Touba, N.A., and E.J. McCluskey, "Test Point Insertion Based on Path Tracing," *Proc. of VLSI Test Symposium*, pp. 2-8, 1996.

[Touba 00] Touba, N. and E.J. McCluskey, "Altering Bit Sequence to Contain Predetermined Patterns," US Patent 6,061,818, May, 2000.

[Wunderlich 90] Wunderlich, H.-J., "Multiple Distributions for Biased Random Test Patterns," *IEEE Transactions on Computer-Aided Design,* Vol. 9, No. 6, pp.584-593, Jun. 1990.